**W02 Learning Activities: CSS Media Queries**

**Overview**

CSS media queries let you apply different styles to your website based on the characteristics of the device being used to view your site. They allow the website to adapt to different screen sizes, orientations, and even user preferences.

Associated Course Learning Outcome(s)

**Prepare**

CSS media queries are essential to responsive web design. The **@media** at-rule specifies condition(s) that determine when a block of CSS rules should be applied. This allows selected elements to be repositioned, resized, hidden, exposed, etc. based on the user's viewport size.

Here is the general syntax for setting up a CSS media query:

@media not|only mediatype and (expressions) {

*/\* CSS rules go here inside the @media query's opening and closing curly brackets {} \*/*

}

Here is an example CSS media query:

1️⃣ @media screen and (min-width: 640px) {

2️⃣ h1 {

3️⃣ font-size: 2.5rem;

4️⃣ margin: 1rem;

5️⃣ color: navy;

6️⃣ }

7️⃣ }

**Code explanation**

1️⃣ **@media screen and (min-width: 640px) {** is the signature line of the @media query block that applies CSS rules when the specified conditions are met. In this example, the conditions require the media type to be a screen and the viewport width is set to be at least 640 pixels.

2️⃣ **h1 {** **<h1> element selector** starts the defined **CSS rule**.

3️⃣-5️⃣ **@font-size: 2.5rem; margin: 1rem; color: navy;** are **declarations**to be applied to any <h1> elements if the **@media** query conditions are met.

6️⃣ The first **}** closes the CSS rule for **<h1>** code elements.

7️⃣ The last **}** closes the media query.

Note that curly brackets { } are used to **contain** a specific media query *and* are also used to define CSS Rules. A common mistake is to close the **@media** query container too soon or not at all. Using the automatic indentation feature of VS Code document formatting features will help you recognize issues with your CSS syntax.

Demonstration Video: ▶️ [Media Queries](https://video.byui.edu/media/t/0_oacos9ak) – [ 2:54 minutes ]

**Activity Instructions**

For this activity, you will create a simple HTML page with two CSS files. You will use CSS media queries to apply the appropriate CSS file based upon the viewport width.

A responsible use of an AI generative tool could be to ask questions about how to formulate a specific code piece. For example, "how to combine selectors in CSS", and an example will be given with which you can apply to your own code.

**File and Folder Setup**

1. Add a new HTML file named "**media-query.html**" in a "**week02**" folder in your **wdd131** repository folder.
2. Add two CSS files named "**media-query.css**" and "**media-query-large.css**" to a subfolder named "**styles**" within the **week02** folder.

**HTML**

1. In your **media-query.html** file, create a valid HTML page with standard **head** content including
   * Meta Charset Attribute
   * Meta Viewport Element
   * Title Element
   * Meta Description Element
   * Meta Author Element
   * Link to a Google Font named "Roboto" using the contemporary code embed provided by fonts.google.com
   * Link references to your two CSS files.

Example

1. In the **body** of the HTML document, add a **header** with an **h1**, a **main** element with two **section** elements, and a **footer** element.
   * The h1 element should contain the words "Media Query Example".
   * Each section h2 heading contains a scripture with book, chapter, and verse.
   * The section paragraph contains a copy of the actual scripture referenced in the heading.
   * The footer should contain your name.

Example

**CSS**

1. Style the document as shown in the example screenshots given below.
   1. **media-query.css**
      1. Do **not** include a media query.
      2. Use the Google Font – Roboto in the **body** rule.
      3. The **header**, **main**, and **footer**
         1. Each have a maximum width of 640 pixels
         2. are centered on the page using **margin: 1rem auto**
         3. include a faint border
         4. use appropriate padding
         5. have a blueish background color of your choice

Remember that you can combine these three selectors into one rule or create a class that contains these common CSS declarations.

* + 1. Declare the main element to be a CSS grid with a grid template of only one column, and a equal gap of 1rem.

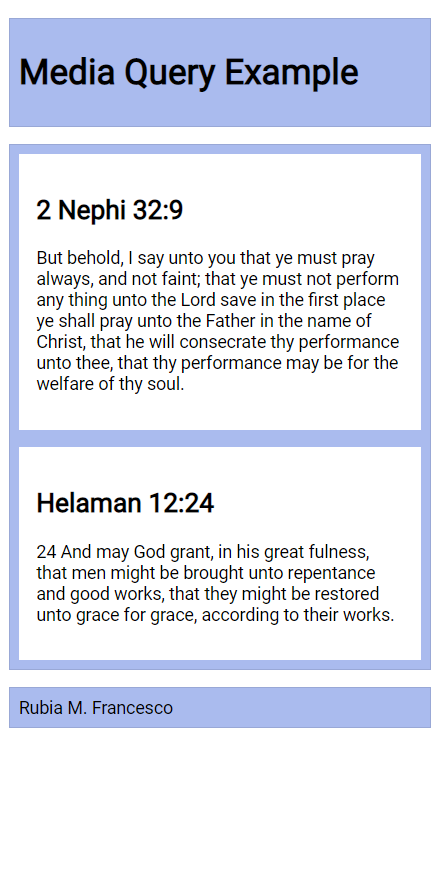
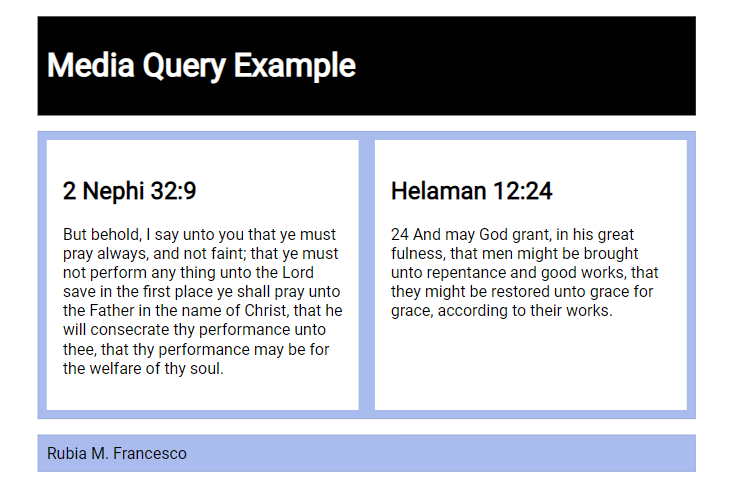
If you had a question to pose to the class in Microsoft Teams, use specific and descriptive language. For example, "*I have a question about the media-query assignment part 5.1.4. I do not know how to set the main element grid to only one column with an even gap of 1rem.*"

* + 1. Declare the section elements to also have padding and a lighter or white background.

Check Your Understanding

1. In the **media-query-large.css:**
   1. Write a containing media query to be applied at a viewport width of 500px or greater.Check Your Understanding
   2. Change through declaration within the media query the **header** to a black background with white text
   3. Change the **main** element to display two columns of equal size.Check Your Understanding

**Example Screenshots**

Phone View ScreenshotWider View ScreenshotCheck Your Understanding

**Setting Breakpoints**

Breakpoints should be set to support the design and content of the page. There are no set breakpoints to use.

"Do NOT define breakpoints based on device classes. Defining breakpoints based on specific devices, products, brand names, or operating systems that are in use today can result in a maintenance nightmare. Instead, the content itself should determine how the layout adjusts to its container."

[How to choose breakpoints – Responsive Web](https://web.dev/articles/responsive-web-design-basics#breakpoints) – Google Web Fundamentals

**Testing**

You should be continuously checking your browser rendered work using Live/Five Server extension.

1. Test your work by resizing your browser window or by using [DevTools device settings](https://developer.chrome.com/docs/devtools/device-mode/) to see the changes in the layout.
2. Use the browser's DevTool [CSS Overview](https://learn.microsoft.com/en-us/microsoft-edge/devtools-guide-chromium/css/css-overview-tool) to check color contrast and other CSS properties.
3. You will need to commit and push your work to your GitHub Pages enabled wdd131 site in order to use the [Page Audit Tool](https://byui-cse.github.io/wdd131-ww-course/audit/w02-media-queries.html).

**W02 Learning Activity: JavaScript Constructs**

**Overview**

Through the prerequisite programming courses, you have built a foundational understanding of common, programming constructs, including operators, expressions, decision structures, loops, and functions. This activity focuses on a few of the common control structures in programming.

Associated Course Learning Outcomes

**Prepare**

Below are some of the most common controls structures, or building blocks, in JavaScript:

Please note that this is not an exhaustive list of JavaScript programming, and you will learn more about these constructs and others throughout the course.

* **Conditional Statements**:
  + **if** statement: Executes a block of code if a specified condition is true.
  + if (condition) {
  + *// Code to execute if the condition is true*

}

The condition is evaluated to a **boolean** value, i.e., true or false. Writing condition statements is a critical skill in programming. The use of [operators](https://byui-cse.github.io/wdd131-ww-course/resources/javascript-operators.html) and expressions is a key concept in writing condition statements that solve programming problems.

* + **else** statement: Provides an alternative block of code to execute if the condition in the if statement is false.
  + if (condition) {
  + *// Code to execute if the condition is true*
  + } else {
  + *// Code to execute if the condition is false*

}

* + **else if** statement: This structure allows for the checking of multiple conditions in sequence.
  + if (condition1) {
  + *// Code to execute if condition1 is true*
  + } else if (condition2) {
  + *// Code to execute if condition2 is true*
  + } else {
  + *// Code to execute if none of the conditions are true*

}

* **Switch Statements**: Provides a way to execute selective blocks of code based on the value of an expression.
* switch (expression) {
* case value1:
* *// Code to execute if expression is equal to value1*
* break;
* case value2:
* *// Code to execute if expression is equal to value2*
* break;
* *// ... more cases ...*
* default:
* *// Code to execute if none of the cases match*

}

* **Looping Statements**:
  + **for** Loop: Repeats a block of coe a specified number of times.
  + for (let i = 0; i < 19; i++) {
  + *// Code to execute in each iteration*

}

* + **while** Loop: Repeats a block of code as long as a specified condition is true.
  + while (condition) {
  + *// Code to execute while the condition is true*

}

* + **forEach** Loop: This loop structure is for **arrays**. It iterates over each element of the array.
  + array.forEach(function(element) {
  + *// Code to execute for each element*

});

**W02 Learning Activity: Document Object Model**

**Overview**

A key skill for any frontend web developer is the ability to manipulate DOM ([DOM](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)), a JavaScript object that is created after the browser initially renders the document. Manipulating the DOM means to read, edit, update, or delete the document, including CSS properties, dynamically. Essentially, the DOM is the outline of the HTML and content nodes.

The purpose of this activity is to introduce the HTML DOM and to learn how to manipulate the document using JavaScript.

Associated Course Learning Outcomes

**Prepare**

Video Overview: [The JavaScript DOM explained in 5 minutes!](https://youtu.be/NO5kUNxGIu0?si=0AV68_oBlm7VnWg3) – Bro Code

* Reference: [Manipulating documents](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents) – MDN.

Note that the [Active learning: A dynamic shopping list](https://mdn.github.io/learning-area/javascript/apis/document-manipulation/shopping-list-finished.html) example found at the end of the MDN article will **help** you complete this week's activities.

* Practice: **Some common DOM manipulation concepts/use cases**.
  + To **select** an HTML element from the document using the [querySelector](https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector) method.  
    This line of code selects the first instance of an article element from the document and assigns the element as a reference to the variable named **article**.

const article = document.querySelector('article');

* + To **change** the innerHTML property of an existing element.  
    This line of code uses an existing variable that has already selected an element and changes its innerHTML property value.

article.innerHTML = 'innerHTML supports <strong>HTML</strong> tags. The textContent property does not.';

* + To **change** the inline CSS style of an element.  
    This line of code changes the text-align CSS property of the selected element.

article.style.textAlign = 'right';

* + To **change** an attribute of an element.  
    This line of code adds and attribute class to the element and gives it a value.

article.setAttribute('class', 'active');

An alternative way to change an attribute of an element—specifically the class attribute—is by directly manipulating the element's **classList**.

articleElement.classList.add('active');

This method is often preferred when working with class manipulation because classList provides additional methods like add, remove, toggle, and contains, making it more convenient and expressive for managing classes on an element.

* + To **create** an element.  
    This line of code creates a new <p> element and stores it in the variable named paragraph.

const paragraph = document.createElement('p');

* + To **append** an element with additional content or elements.  
    These lines of code add content to the end of the article element.
  + article.appendChild(paragraph);

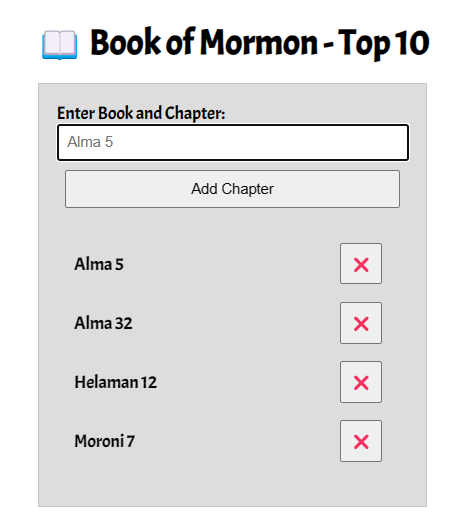
article.append(paragraph, 'Hello World Addition!');

The append() method allows you to include multiple arguments to append to the element in the specified order.

* + To **remove** an element from the document.  
    These lines of code remove the paragraph from the article and then, the article itself.
  + article.removeChild(paragraph);

article.remove();

**Activity Instructions**



This Book of Mormon application will be added upon in future learning activities. We start now with the interface and basic DOM assignments and build.

This app allows users to enter their favorite Book of Mormon chapters and display them on a list that is updated automatically on the screen. Entries can then be deleted from the displayed list.

1. Create an HTML file named "**bom.html**" in the **week02** folder.
2. Your **bom.html** HTML document should include the basic meta tags and an appropriate title.
3. Create an external CSS file and a JS file and place them in appropriate subfolders within the week02 folder.Check Your Understanding
4. Copy and paste the basic interface (the HTML and CSS) from the following CodePen ☼ [BOM Top 10](https://codepen.io/BYU-Idaho/pen/QwwKMzL).
5. In your blank js file, declare three (3) variables that hold references to the **input**, **button**, and **list** elements.Check Your Understanding
6. Create a **li** element that will hold each entry's chapter title and an associated delete button.Check Your Understanding
7. Create a delete **button**.Check Your Understanding
8. Populate the **li** element variable's **textContent** or **innerHTML** with the input value.Check Your Understanding
9. Populate the button textContent with a ❌.Check Your Understanding
10. Append the **li** element variable with the delete button.Check Your Understanding
11. Append the **li** element variable to the unordered list in your HTML.Check Your Understanding

So far, You have setup the interface and completed some basic DOM interaction. The application **will not work at this point**. The next activity will teach how to respond to events, like button clicks. You also need to wait for the user to make an entry into the favorite chapter text field before processing.

You need to consider screen readers and how they will interpret anything in the content. For example, the delete button just has an emoticon and may not read correctly as the button to remove a chapter. What can we do? One solution is to create a [aria-label attribute](https://developer.mozilla.org/en-US/docs/Web/API/Element/ariaLabel) on the button with a value like "Remove Alma 5".

**Submission**

**W02 Learning Activity: Handling DOM Events**

**Overview**

Handling DOM events in JavaScript is a fundamental part of building interactive web pages. In this activity, you'll learn all about the different events you can handle, how to handle them, and how to pass data between the event handler and the rest of your code.

"Events are things that happen in the system you are programming – the system produces (or "fires") a signal of some kind when an event occurs, and provides a mechanism by which an action can be automatically taken (that is, some code running) when the event occurs. Events are fired inside the browser window, and tend to be attached to a specific item that resides in it. " – MDN

**Prepare**

1. Reference: [Introduction to Events](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events) – MDN
2. Practice: **Some common DOM event concepts/use cases**. These are just a few examples, and there are many other events and use cases.
   * **Click Event**: Triggered when a user clicks on an element. It's widely used for handling button clicks, link clicks, or any interaction involving a mouse click.
   * buttonElement.addEventListener('click', function() {
   * *// Code to execute when the element is clicked*

});

The **addEventListener** method takes two arguments: the event name and a function to execute when the event is triggered. The function is called an event handler. We will learn more about writing functions later in the course. For now, just know that the function is a block of code that will be executed when the event is triggered. The function is also called a callback function because it is called back when the event is triggered.

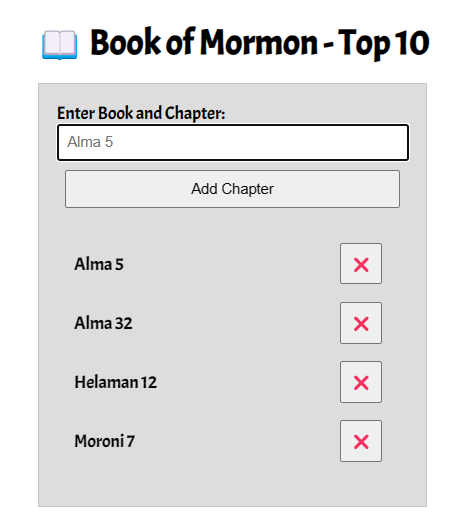
* + **Keyup Event**: Triggered when a key is released. Useful for handling keyboard input.
  + buttonElement.addEventListener('keyup', function() {
  + *// Code to execute when a key is released*

});

* + **DOMContentLoaded Event**: Triggered when the HTML document has been completely parsed and all deferred scripts have been executed. It's widely used for initializing JavaScript applications.
  + document.addEventListener("DOMContentLoaded", function() {
  + *// Code to execute when the DOM is parsed*

});

**Activity Instructions**



This activity adds functionality to the BOM Top 10 application started in the [previous learning activity on the DOM](https://byui-cse.github.io/wdd131-ww-course/week02/prepare-dom-manipulation.html). We will organize the code to respond to the **Add Chapter** button click in order to build the list item. In addition, the delete button functionality will be added along with some user interface expectations.

1. Open up your JavaScript file that is supporting the **bom.html** application if needed.
2. Create a **click** event listener for the **Add Chapter** button using an **addEventListener**.Check Your Understanding
3. Within the **Add Chapter** button click event function block, between the opening and closing of the callback function argument { ... }, do the following:
   * Check to make sure the input is not blank before completing the remaining tasks in this list by using an **if** block that provides a message or does nothing and returns the **.focus()** to the input field.Check Your Understanding
   * Move the code that you wrote in the last activity into this **if** block in order to support the correct flow of the program.Check Your Understanding
   * Add an event listener to the delete button that removes the li element when clicked.Check Your Understanding

This event listener function is using **Event Delegation** which is an efficient way to handle events on multiple elements (the BOM chapters). Instead of adding an event listener to each individual delete button, you can add a single event listener to the parent element (the list) and use the **e.target** property to determine which delete button was clicked. This is especially useful when you have a large number of elements or when elements are dynamically added or removed from the DOM.

* + Change the input value to nothing or the empty string to clean up the interface for the user.Check Your Understanding
  + Whether or not a list item was created, the focus (active cursor) should be sent to the input element.Check Your Understanding

**Testing & Submission**

* Thoroughly test your application in the browser. This can be done locally.
* What else would you add to increase this applications usability?
* Commit and push your work to your GitHub Pages enabled wdd131 repositor

**W02 Learning Activity: Responsive Menus**

**Overview**

The purpose of this activity is to build a responsive menu using JavaScript. The menu will adapt to different viewport sizes and also respond when the user clicks on a "hamburger" button (≡). In responsive design, our objective is to provide users with a positive and familiar experience.

Associated Course Learning Outcome(s)

**Prepare**

Hamburger menus or equivalent structures support responsive design. They allow us to clean up layouts because they save space on the page. They are well-recognized by users, even if they may not know that the common symbol is called a hamburger menu.

Here is an example from Google Maps.



1. Watch this example application of a responsive menu in this video demonstration:
   * Video Demonstration: ▶️ [Responsive Hamburger Menu](https://video.byui.edu/media/t/1_uzrqq84w) – [ 6:14 minutes ]
   * CodePen ☼ [Responsive Menu](https://codepen.io/BYU-Idaho/pen/yyBwGdO)

**Activity Instructions**

1. In your own CodePen account, this CodePen ☼ [Responsive Menu – Start](https://codepen.io/BYU-Idaho/pen/GgKezzJ)
2. **CSS**: Move the hamburger menu button to the right upper corner of the screen by supplying appropriate CSS declarations where there are currently blanks (\_\_\_\_\_\_).
3. **JavaScript**: Complete the JavaScript code by filling in the three (3) blanks (\_\_\_\_\_\_) within the menu button event listener.

This example listens to a user initiated event and toggles the given class for the navigation element. The class name already exists in CSS and is ready to be applied.

1. **CSS**: Change the hamburger button closing icon ❎ to something of your choice that would be appropriate.What does the **::** syntax in the CSS rule selector do?
2. **CSS**: **Wayfinding** – Add a rule for the "active" class to let the user know what page they currently have open by visually changing the appearance of the one menu item.  
   In this example the **active** class is being applied to the About page.

<a href="#" title="About Us" class="active">About Us</a>

"Making it easy for people to navigate around a website or application helps everyone find what they need quickly and effectively. Clear wayfinding especially helps people with visual, mobility, or cognitive impairments who may otherwise find it difficult to understand where they are and how to get where they want to go." – [Harvard University – Digital Accessibility](https://accessibility.huit.harvard.edu/support-navigation-and-wayfinding)

1. Be sure to **Save** your changes on your CodePen.

**02 Learning Activity: JavaScript Debugging**

**Overview**

Debugging is the process of finding and fixing errors in your code. This is an essential skill for any developer. You will learn how to use the browser's DevTools to debug your JavaScript code.

**Prepare**

**Types of Errors**

There are three types of errors that you will encounter when writing JavaScript code:

* **Syntax Errors** – These are errors that occur when you write code that does not follow the rules of the JavaScript language. These errors are caught by the browser and will prevent your code from running.
* **Runtime Errors** – These are errors that occur while your code is running. These errors can be caused by a variety of issues, such as trying to access a property of an object that does not exist or calling a function that does not exist.
* **Logical Errors** – These are errors that occur when your code does not produce the expected result. These errors can be difficult to find because they do not cause the browser to throw an error.

**Activity Instructions**

This activity focuses on debugging techniques for the browser.

**Setup Files**

1. In your **week02** subfolder, add a file named "**debugging.html**".
2. In the **debugging.html** file, type **!** and then press the **tab** key on your keyboard. This action will automatically add the basic components of an HTML document.
3. Add a subfolder named "**scripts**" to the **week02** folder, add a file named "**debugging.js**".
4. In **debugging.html** file, add a **script** reference to the **debugging.js** JavaScript file.Check Your Understanding

**HTML**

1. Add the following elements to the HTML **body** of your **debugging.html** page:
2. <h1>JavaScript Debugging using DevTools</h1>

<p>Area of circle with radius <span id="radius"></span>: <span id="area"></span></p>

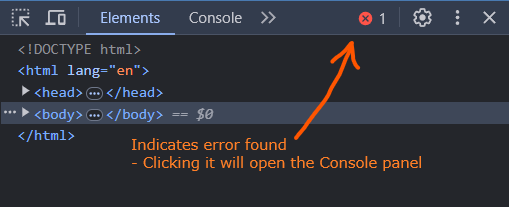
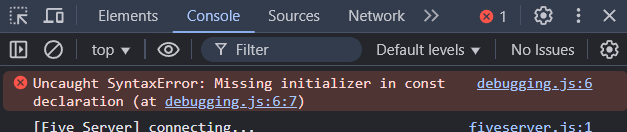
**JavaScript**

1. Add the following JavaScript statements to your **debugging.js** file:
2. const radiusOutput = document.getElementById('radius');
3. const areaOutput = document.querySelector('area');
4. let area = 0;
5. const PI == 3.14159;
6. const radius = 10;
7. area = PI \* radius \* radius;
8. radiusOutput = radius;
9. areaOutput = area;
10. radius = 20;
11. area = PI \* radius \* radius;
12. radiusOutput = radius;

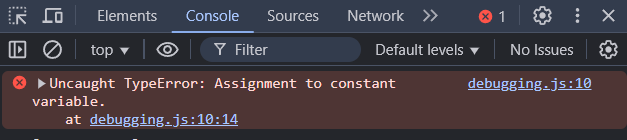
areaOutput = area;

**Debugging**

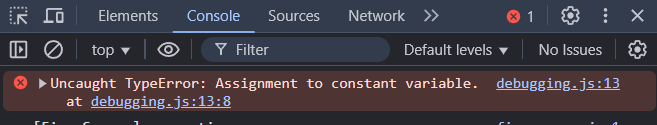
Video Demonstration: ▶️ [Debugging JavaScript using DevTools Activity Walkthrough](https://www.loom.com/share/2b6aa47d5d204d18ba16619200ed6d46?sid=d34b0366-1676-4759-ad5f-98a79ad74709) – [ 5.56 minutes ]

1. Using Live/Five Server, open your **debugging.html** file in a browser.
2. Open the browser's DevTools and open the the **Console** tab.DevTools Console
3. Note any redlined errors in the DevTools console and outlined in Console window. These are syntax or runtime errors. This errors will stop your code from running until they are fixed.
4. Fix the first error outlined in the **debugging.js** file. The image below indicates that the error is on line 6 (**debugging.js:6**)The comparison operator is confusing the compiler. In VS Code, remove the extra equal sign.

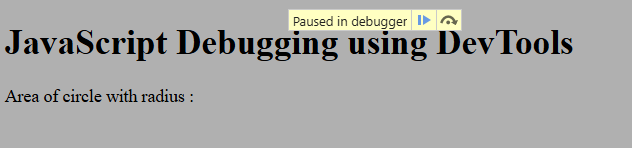
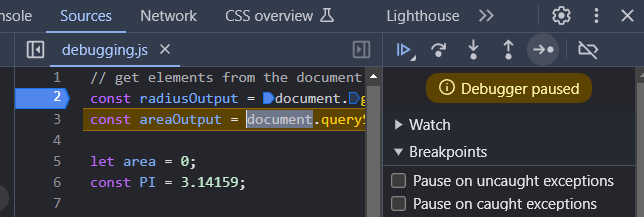
const PI = 3.14159;

1. In your localhost browser session, refresh the page and check the console for the next error.This error is referring to the issue of trying to assign a value to an HTML element. This might be confusing given the error message says "Assignment to constant variable". Change the line of code for both elements to use the **textContent** property.
2. radiusOutput.textContent = radius;

areaOutput.textContent = area;

1. In your localhost browser session, refresh the page and check the console for the next error.This error's message is accurate. The code is attempting to assign a new value to the **radius** variable which was declared as a constant. Change the variable declaration to a **let** instead of **const**.

let radius = 10;

1. Fix any other errors found in the console.
2. After you get output to the page that seems correct, open the **Sources** panel in DevTools.
3. Practice "step debugging" through the JavaScript code by setting a breakpoint – click on the line number where you want to execute a pause in the program.
4. Refresh the page to trigger the code to run again. The body will be grayed out and indicate that the page load was "Paused in debugger"
5. Use the **Step →•** button to move through the code line by line.
6. At any time you can pass your mouse over a variable to view its current assignment value.
7. Continue to step through the code until you have a good understanding of how the code is executing.